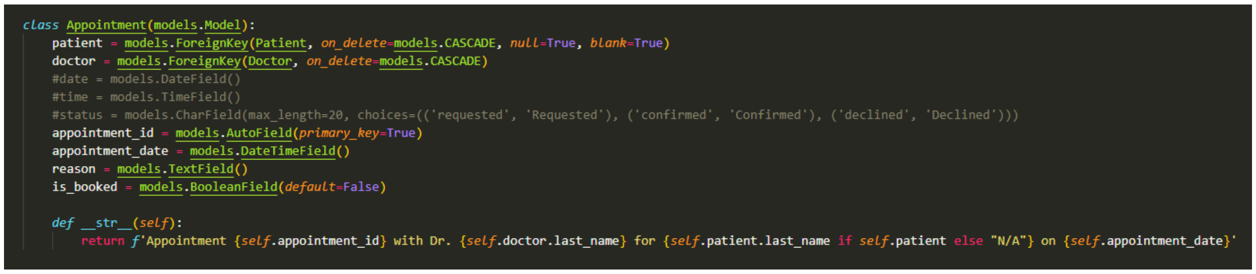
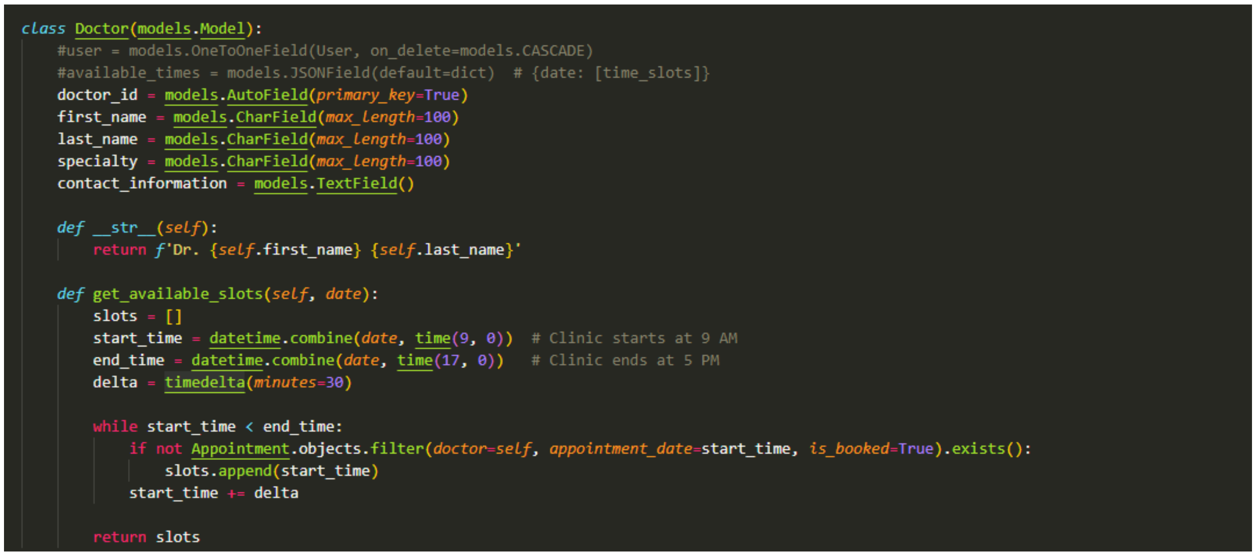
The artifact I’m submitting is the code for the Patient to be able to request an appointment, and to have the system check the doctor’s schedule to see if there’s availability or not.

First thing I did was update the “Appointment” model. As you can see, I commented out what I had before so that you’re able to see what changed and what was added in.



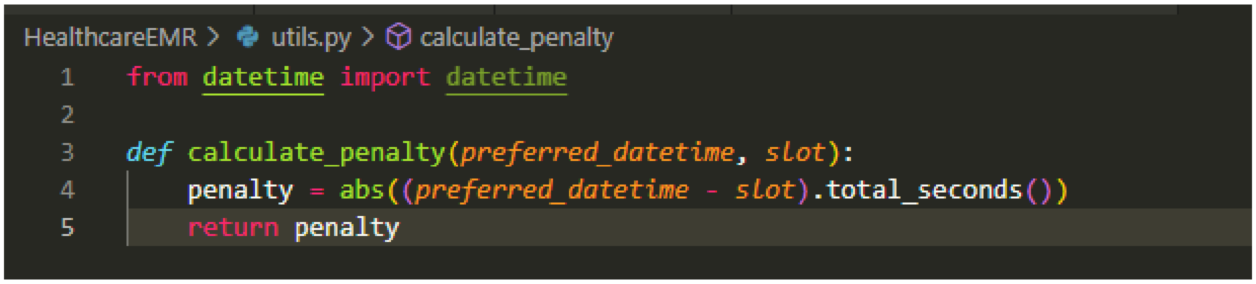
```python
class Appointment(models.Model):
    patient = models.ForeignKey(Patient, on_delete=models.CASCADE, null=True, blank=True)
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE)
    #date = models.DateField()
    #time = models.TimeField()
    #status = models.CharField(max_length=20, choices=(('requested', 'Requested'), ('confirmed', 'Confirmed'), ('declined', 'Declined')))
    appointment_id = models.AutoField(primary_key=True)
    appointment_date = models.DateTimeField()
    reason = models.TextField()
    is_booked = models.BooleanField(default=False)

    def __str__(self):
        return f'Appointment {self.appointment_id} with Dr. {self.doctor.last_name} for {self.patient.last_name if self.patient else "N/A"} on {self.appointment_date}'
```

I then went to go add a method within the “Doctor” model so to retrieve availability time slots for the Doctor:



```python
class Doctor(models.Model):
    #user = models.OneToOneField(User, on_delete=models.CASCADE)
    #available_times = models.JSONField(default=dict)  # {date: [time_slots]}
    doctor_id = models.AutoField(primary_key=True)
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    specialty = models.CharField(max_length=100)
    contact_information = models.TextField()

    def __str__(self):
        return f'Dr. {self.first_name} {self.last_name}'

    def get_available_slots(self, date):
        slots = []
        start_time = datetime.combine(date, time(9, 0))  # Clinic starts at 9 AM
        end_time = datetime.combine(date, time(17, 0))   # Clinic ends at 5 PM
        delta = timedelta(minutes=30)

        while start_time < end_time:
            if not Appointment.objects.filter(doctor=self, appointment_date=start_time, is_booked=True).exists():
                slots.append(start_time)
            start_time += delta

        return slots
```

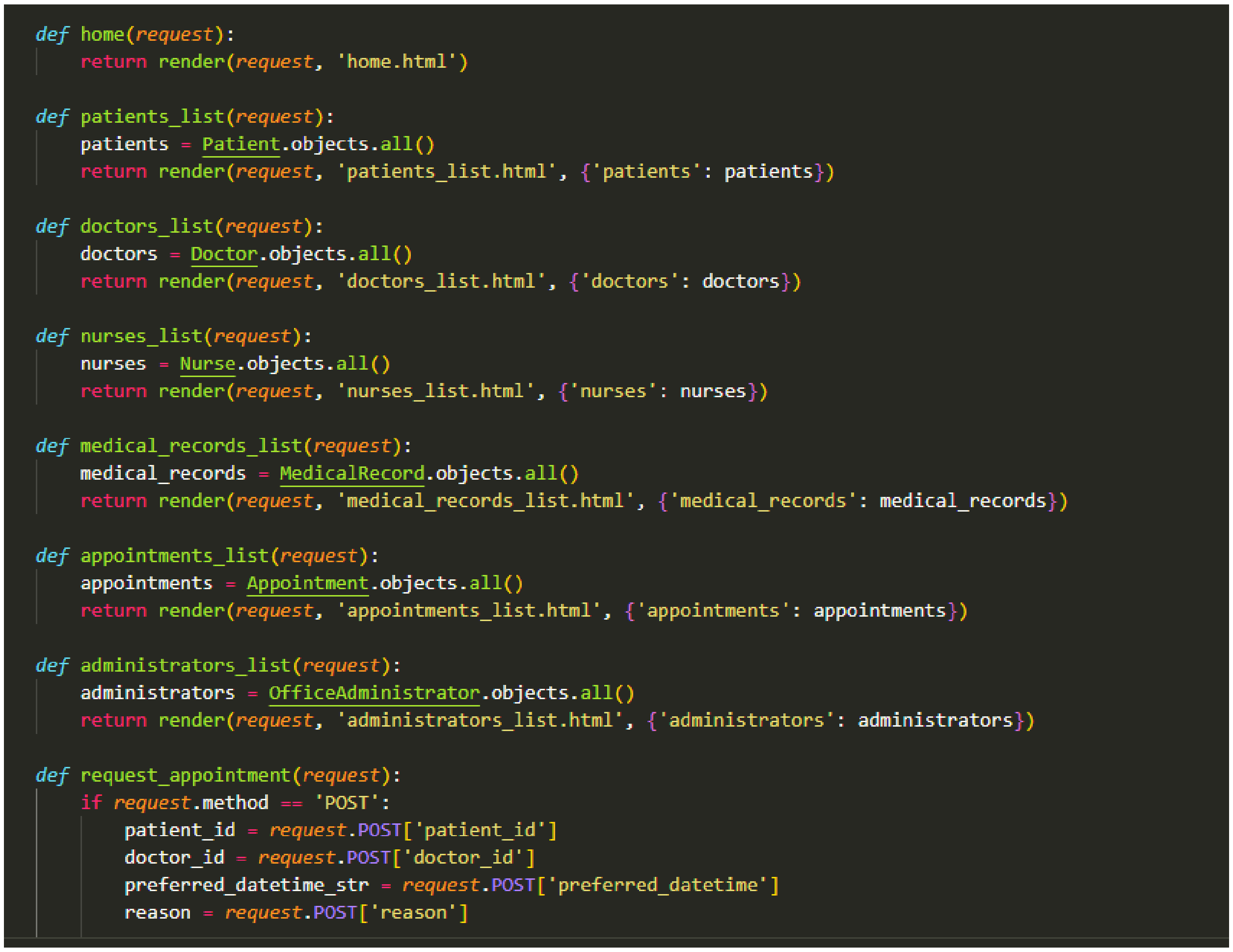
I had to create a file called “utils.py” so that the system could calculate the penalties:



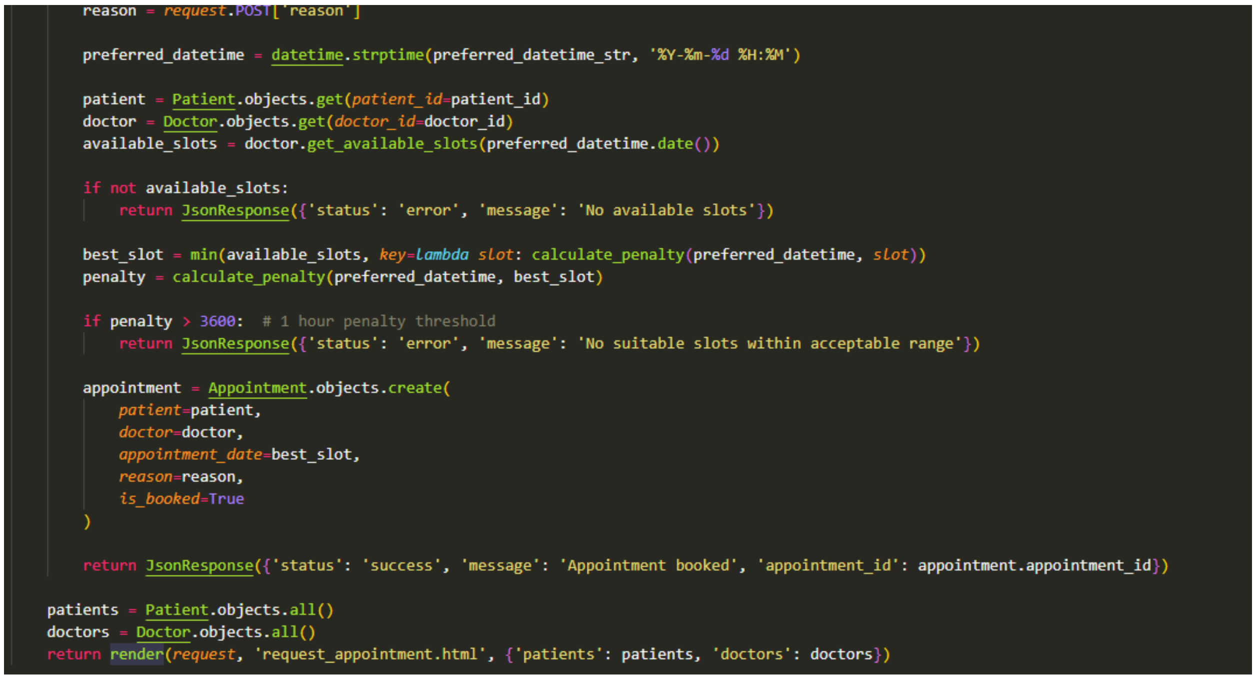
HealthcareEMR > utils.py > calculate_penalty

```python
from datetime import datetime

def calculate_penalty(preferred_datetime, slot):
    penalty = abs((preferred_datetime - slot).total_seconds())
    return penalty
```

After creating the utils file and code, I updated the “Views” app to be able to handle the appointment requests/booking:

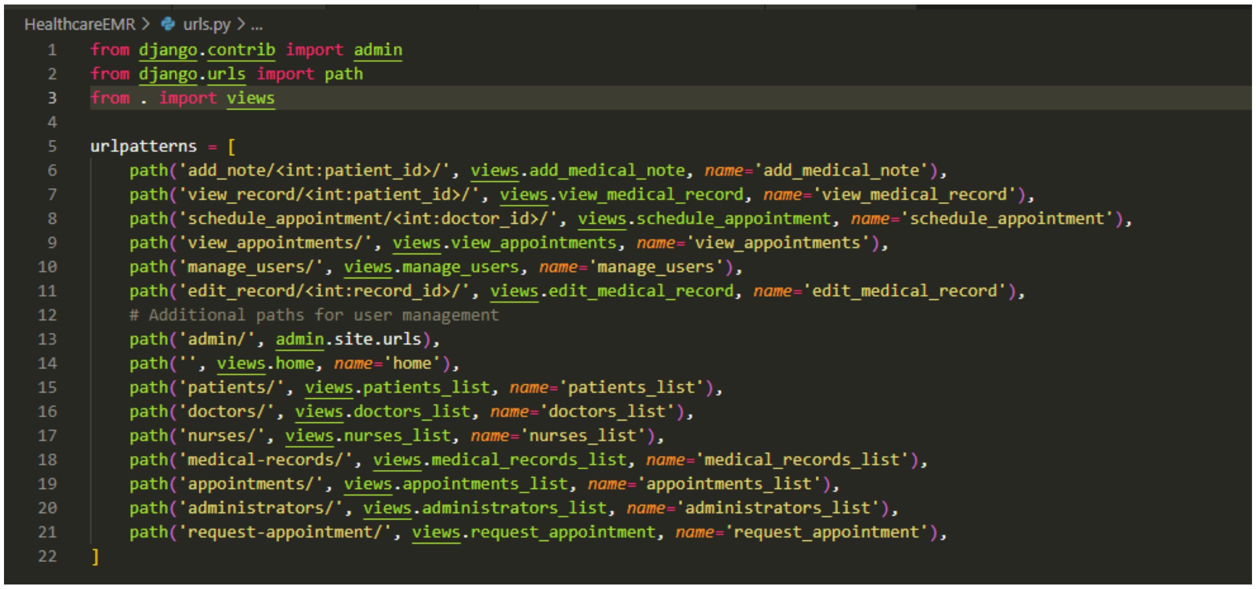


```python
def home(request):
    return render(request, 'home.html')

def patients_list(request):
    patients = Patient.objects.all()
    return render(request, 'patients_list.html', {'patients': patients})

def doctors_list(request):
    doctors = Doctor.objects.all()
    return render(request, 'doctors_list.html', {'doctors': doctors})

def nurses_list(request):
    nurses = Nurse.objects.all()
    return render(request, 'nurses_list.html', {'nurses': nurses})

def medical_records_list(request):
    medical_records = MedicalRecord.objects.all()
    return render(request, 'medical_records_list.html', {'medical_records': medical_records})

def appointments_list(request):
    appointments = Appointment.objects.all()
    return render(request, 'appointments_list.html', {'appointments': appointments})

def administrators_list(request):
    administrators = OfficeAdministrator.objects.all()
    return render(request, 'administrators_list.html', {'administrators': administrators})

def request_appointment(request):
    if request.method == 'POST':
        patient_id = request.POST['patient_id']
        doctor_id = request.POST['doctor_id']
        preferred_datetime_str = request.POST['preferred_datetime']
        reason = request.POST['reason']
```



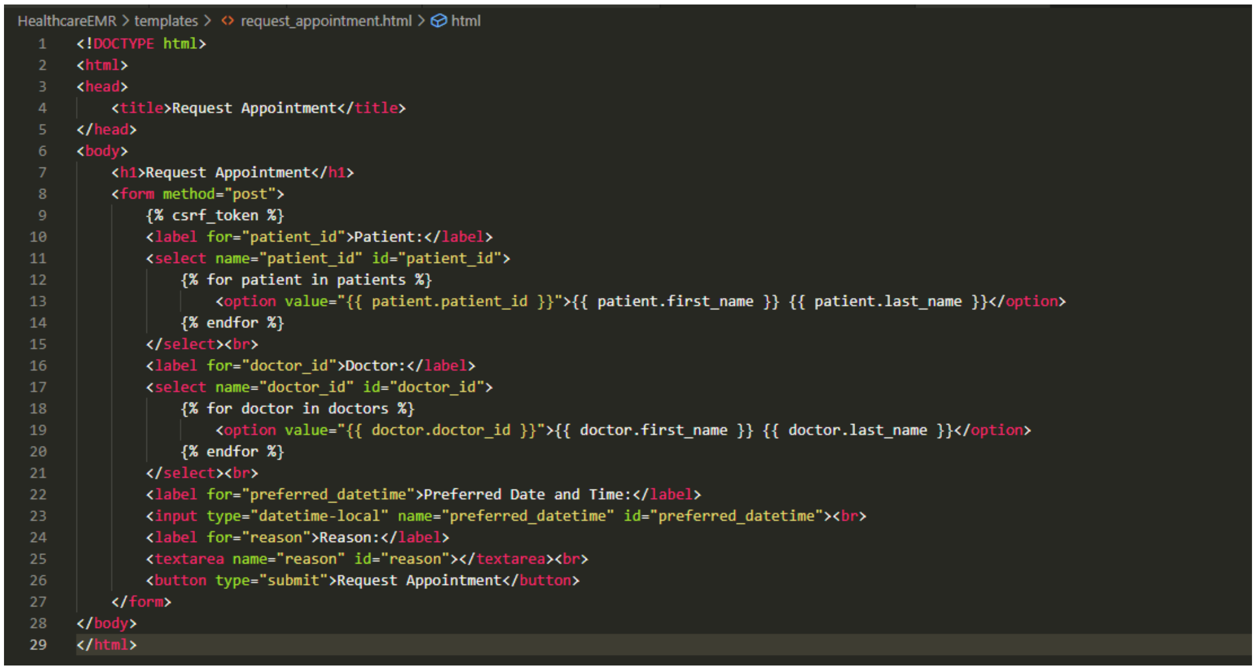
```python
        reason = request.POST['reason']

        preferred_datetime = datetime.strptime(preferred_datetime_str, '%Y-%m-%d %H:%M')

        patient = Patient.objects.get(patient_id=patient_id)
        doctor = Doctor.objects.get(doctor_id=doctor_id)
        available_slots = doctor.get_available_slots(preferred_datetime.date())

        if not available_slots:
            return JsonResponse({'status': 'error', 'message': 'No available slots'})

        best_slot = min(available_slots, key=lambda slot: calculate_penalty(preferred_datetime, slot))
        penalty = calculate_penalty(preferred_datetime, best_slot)

        if penalty > 3600:  # 1 hour penalty threshold
            return JsonResponse({'status': 'error', 'message': 'No suitable slots within acceptable range'})

        appointment = Appointment.objects.create(
            patient=patient,
            doctor=doctor,
            appointment_date=best_slot,
            reason=reason,
            is_booked=True
        )

        return JsonResponse({'status': 'success', 'message': 'Appointment booked', 'appointment_id': appointment.appointment_id})

    patients = Patient.objects.all()
    doctors = Doctor.objects.all()
    return render(request, 'request_appointment.html', {'patients': patients, 'doctors': doctors})
```

Then I added in the URL configurations for the appointment request view:



```python
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('add_note/<int:patient_id>/', views.add_medical_note, name='add_medical_note'),
    path('view_record/<int:patient_id>/', views.view_medical_record, name='view_medical_record'),
    path('schedule_appointment/<int:doctor_id>/', views.schedule_appointment, name='schedule_appointment'),
    path('view_appointments/', views.view_appointments, name='view_appointments'),
    path('manage_users/', views.manage_users, name='manage_users'),
    path('edit_record/<int:record_id>/', views.edit_medical_record, name='edit_medical_record'),
    # Additional paths for user management
    path('admin/', admin.site.urls),
    path('', views.home, name='home'),
    path('patients/', views.patients_list, name='patients_list'),
    path('doctors/', views.doctors_list, name='doctors_list'),
    path('nurses/', views.nurses_list, name='nurses_list'),
    path('medical-records/', views.medical_records_list, name='medical_records_list'),
    path('appointments/', views.appointments_list, name='appointments_list'),
    path('administrators/', views.administrators_list, name='administrators_list'),
    path('request-appointment/', views.request_appointment, name='request_appointment'),
]
```

And finally, I created a “Request Appointment” HTML template:



```html
<!DOCTYPE html>
<html>
<head>
    <title>Request Appointment</title>
</head>
<body>
    <h1>Request Appointment</h1>
    <form method="post">
        {% csrf_token %}
        <label for="patient_id">Patient:</label>
        <select name="patient_id" id="patient_id">
            {% for patient in patients %}
                <option value="{{ patient.patient_id }}">{{ patient.first_name }} {{ patient.last_name }}</option>
            {% endfor %}
        </select><br>
        <label for="doctor_id">Doctor:</label>
        <select name="doctor_id" id="doctor_id">
            {% for doctor in doctors %}
                <option value="{{ doctor.doctor_id }}">{{ doctor.first_name }} {{ doctor.last_name }}</option>
            {% endfor %}
        </select><br>
        <label for="preferred_datetime">Preferred Date and Time:</label>
        <input type="datetime-local" name="preferred_datetime" id="preferred_datetime"><br>
        <label for="reason">Reason:</label>
        <textarea name="reason" id="reason"></textarea><br>
        <button type="submit">Request Appointment</button>
    </form>
</body>
</html>
```

Overall, what this will do is allow the patient to request appointments by their specified date and time and their reasoning. The system will then check the doctor's availability while also calculating penalties for each slot, and then it'll either book an optimal slot or notify the patient that there's not suitable slots available.

I feel that this ensures that I met the course objectives, as I'm showcasing the algorithm for the system to determine an available slot based on the patient's needs and the doctor's availability. This artifact was started on July 24, 2024 and finished on July 28, 2024.

What I learned from this is how it's important to map out the steps (pseudocode & flowchart) for the algorithm to transpire before coding. This made it so much easier to code since I developed the map for it in a previous module (the first one I believe). This also ensures that I'm tending to both the patient and doctor's priorities without the need of overbooking the Doctor but also allowing the patient to schedule an appointment that's not too far out.